Process model patterns for collaborative work

Jacques Lonchamp

PROCESS MODEL PATTERNS FOR COLLABORATIVE WORK

Jacques Lonchamp[1]

Abstract

As most real work is collaborative in nature, process model developers have to model collaborative situations. This paper defines generic collaborative patterns, ie, pragmatic and abstract building blocks for modelling recurrent situations. The first part specifies the graphical notation for the solution description. The second part gives some typical patterns for the collaborative production of a single document in isolation and for the synchronization of two dependent documents. The conclusion emphasizes some implications for process-centred systems.

1. Introduction

Process-centred systems (PCSs), like process-sensitive software engineering environments [8] or workflow management systems [4], are based on the explicit definition of the process they support. As most real work is collaborative in nature, ie, not done individually, but rather in groups, process model developers have to model collaborative situations. An obvious solution for reducing the process modelling effort is to provide process model developers with generic collaboration patterns, defining basic building-blocks for constructing their specific models. This paper gives some preliminary results about this issue.

The term 'pattern' has the meaning given initially by C. Alexander for architectural patterns [1]: « each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way you can use this solution a million times over, without ever doing it the same way twice ». In other terms, a pattern is an abstract solution to a problem in a context. As for OO design patterns [9], the granularity is neither too small (« not about design that can be encoded in classes »), nor too large (« not complex, domain specific designs for an entire application or subsystem »). In addition, the term 'collaboration' (or 'collaborative') is used as the umbrella-term for all kinds of collective work, mixing communication aspects, coordination aspects, and co-decision aspects.

The second section describes how the problem is tackled in some related approaches, and contrasts them with the view taken in this paper. The third section summarizes the notation used for describing the pattern solution part. The fourth section illustrates the approach through a selection of typical patterns. Finally, the conclusion discusses some implications for PCSs.

[1] LORIA-Université Nancy 2, BP 239, 54506 Vandoeuvre-lès-Nancy Cedex, France (e-mail: jloncham@loria.fr)

2. Related approaches

Many different kinds of building blocks are described in related approaches. We summarize here three of them. However, a majority of PCSs give only low level constructs and process model developers have to build their models of collaborative situations from scratch.

2.1. Conversation-based workflow management systems

In this first category, an activity always results from the request from one actor (the requester or customer) to another actor (the assignee or performer). For instance, in Action Workflow [13] a generic loop is the basic component of all process models (see Figure 1).

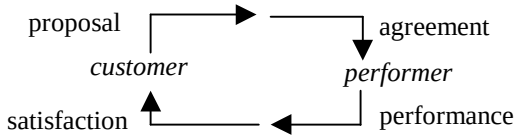


Figure 1. Action Workflow loop.

A loop can complete any quadrant of another loop. Process models are networks of related loops. In Regatta [16], the assignee can choose among three different ways of performing the activity: manually, by instantiating a plan template, or by creating a new plan from scratch. A plan is a network of stages, each stage involving in turn a loop between a requester and an assignee.

2.2. Domain specific process modelling languages

In this second category, systems provide process modelling languages dedicated to a specific class of collaborative applications. For instance, there exist several proposals of PCSs dedicated to collaborative review/inspection applications [12, 17]. The process developer just specifies in the specific language his/her choices for customizing the whole collaborative application.

2.3. Transactional systems

In this third category, approaches emphasize consistency and integrity maintenance of shared documents. Classically consistency and correctness are asserted by isolating concurrent activities (ACID transactions). Exchanges are only possible at the start/end of activities. But exchange of intermediate results is necessary for collaborating. Advanced transaction models are required to favour cooperation while continuing to assert some correctness properties [5]. These models provide predefined strategies, associated with privileged cooperation structures.

2.4. Discussion

In conversation-based approaches of section 2.1, the generic loop structure implies a very specific way for structuring and deploying the process model, having an impact on both the ‘production’ process and the ‘meta’ process [2]. The plan template concept of Regatta is a specific solution for a given activity, and not a generic reusable building block. The last remark also applies for the kind of reuse provided by domain specific approaches of section 2.2. In this case, the analogy is more with OO frameworks [9] than with OO design patterns. Transactional systems are based on some

theoretical correctness criterion which is implemented into some predefined strategy. Unfortunately, there exist many different notions of correction, and each extended transactional model has a limited applicability and a high level of rigidity. In summary, and by contrast, collaboration patterns considered in this paper specify pragmatic solutions to recurrent problems: they provide basic, generic, and abstract building blocks for constructing 'production' process models.

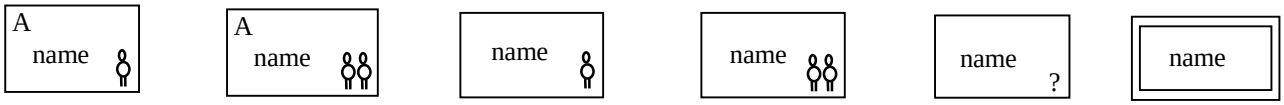
3. The notation for the pattern solution part

The solutions are expressed with a notation designed for showing visually their most important aspects. It has some similarities with the APEL visual process modelling language [6]. The control part is similar to many workflow modelling notations [4, 18, 19], and its semantics is specified through (free choice) Petri nets. The control part shows the task ordering. The data part shows how product artifacts flow between tasks or are shared by them.

3.1. Tasks

A process model is a network of tasks. Tasks are either atomic or can be refined into a network of sub-tasks. A workspace is associated to each atomic task, where actors perform their work. A workspace provides actors with the products and tools they need, in a right form and a right location (computer). Workspaces can exchange product artifacts and can also share them, if there is some common repository. A task (of any kind) is depicted by a rectangular box. Atomic tasks have an 'A' letter in the top left-hand corner, which distinguishes them from compound tasks. A single man icon in the bottom right-hand corner depicts an individual task, while a group icon (with two men) depicts a collective task. A compound task is refined statically into a network of sub-tasks. A compound individual task is refined exclusively into individual sub-tasks. A compound collective task is refined into either individual or collective sub-tasks. A question mark depicts an undefined task, ie, a task that must be refined 'on the fly', during process execution. A multiple task, with a double borderline, has at least one instance: its exact number of instances is only known at run time.

Figure 2. Graphic representation of task types.



Atomic individual | Atomic collective | Compound individual | Compound collective | Undefined | Multiple

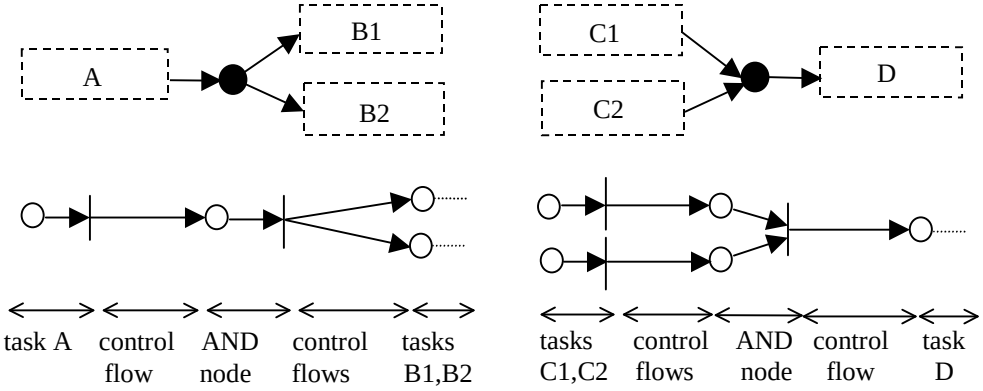
3.2. Control flows

Flows of control between tasks are depicted by solid arrows. In terms of Petri nets, a place is associated to each task, followed by as many transitions as there are successors (ie, destinations of outgoing arrows). Each control flow has a corresponding edge from a transition to a place.

Figure 3.

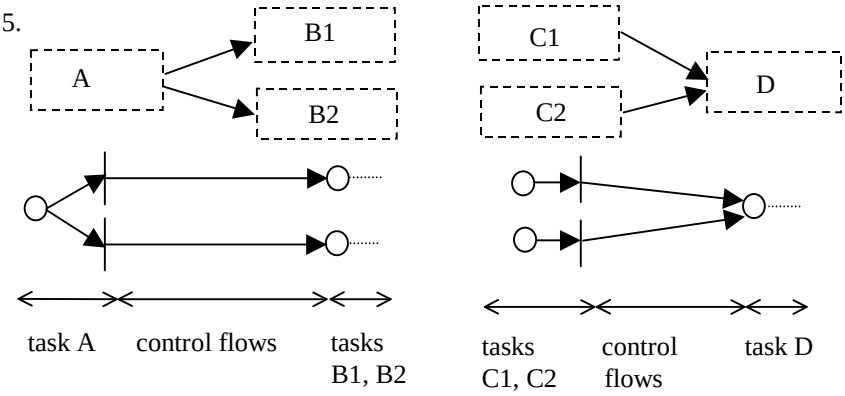
A black circle models an AND node, useful for describing a parallel split of the flow of control, or a join between flows of control. A transition, preceded by as many places as there are predecessors (ie origins of incoming arrows), is associated to each AND node.

Figure 4.



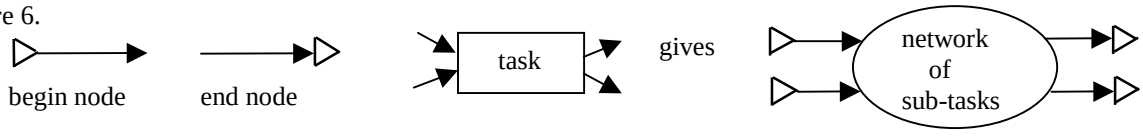
If a task has several outgoing arrows, it models an OR split. If a task has several incoming arrows it models an OR join.

Figure 5.



White triangles model either begin or end nodes. If a task is refined into a network of sub-tasks, its incoming (resp. outgoing) arrows become internal begin (resp. end) nodes.

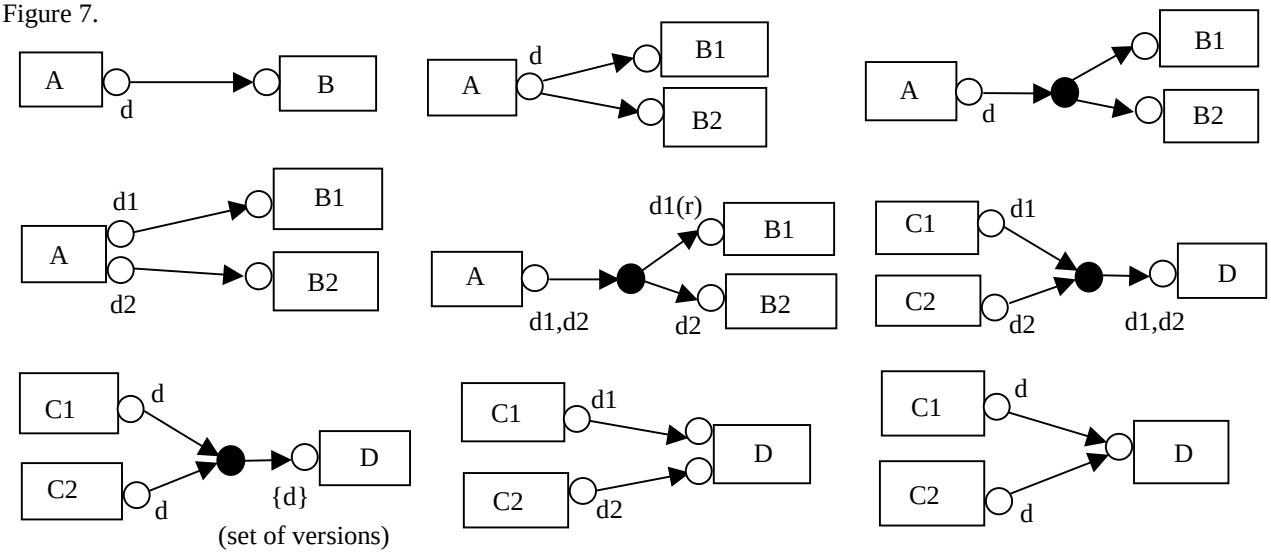
Figure 6.



3.3. Data flows and data sharing

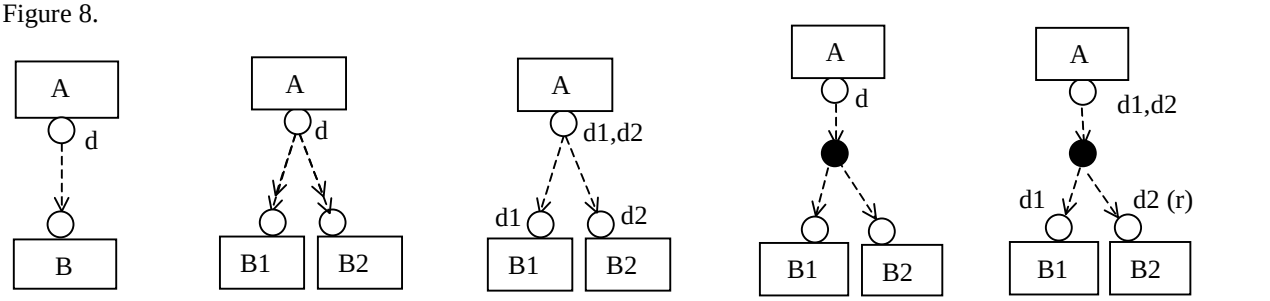
Product artifacts are modelled as small circles. They can flow between tasks or be shared by them.
a) A synchronous data flow (SDF) takes place when the sender task terminates and before the receiver task starts. It is a flow of final results between ‘transactional workspaces’ [6]. The product is removed from the sender and transferred to the receiver. It is represented in conjunction with the corresponding control flow, always on the vertical sides of tasks, because by convention the time flows from the left to the right: the right side corresponds to the end of the task and the left side corresponds to its beginning. The additional notation d(r) specifies that d is transferred in read-only mode. Figure 7 shows different examples of SDFs.

Figure 7.



b) An asynchronous data flow (ADF) takes place during task execution. It is a flow of intermediate results between 'cooperative workspaces' [6]. It is represented by a dotted arrow always connected to the horizontal sides of the concurrent tasks (ie, between their beginning and their end). The product is either copied (simple arrow) or transferred (double arrow). When the product is copied, changes are performed separately by the tasks. There can exist many strategies for deciding when products are sent or fetched. Figure 8 shows different examples of ADFs.

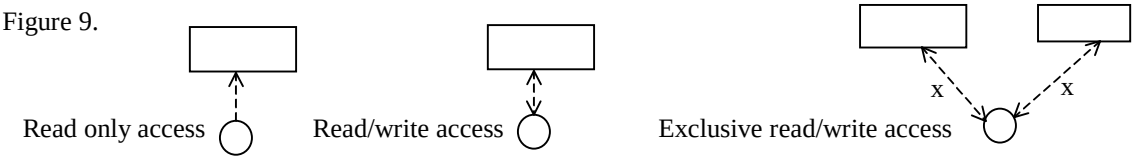
Figure 8.

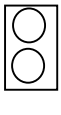


d is copied into B | d is transferred into B1 or B2 | d1 is copied into B1 or d2 is copied into B2 | d is copied into B1 and B2 | d1 is copied into B1 and d2 is copied into B2 (read only)

c) Product sharing is possible if there exists some common repository. In this case, products are not depicted close to tasks. A read only access is shown by a dotted arrow from the product. A read/write access is depicted by a two sides dotted arrow. Exclusive read/write access is specified by the 'x' letters on the corresponding arrows.

Figure 9.



Shared product artifacts may be versionned. When necessary, it is possible to distinguish between purely linear versionning and versionning with alternative paths, respectively with  and  .

4. Some typical collaboration patterns

We describe in this section several patterns, which cover some of the most important collaborative situations, but certainly not all of them. These patterns result from pragmatic studies of many processes.

4.1. The ‘co-work pattern’

Problem Collaborative production of a single document in isolation.
Context Several actors work together within the same atomic task. There exist many ways to perform such co-work: interaction can either be synchronous or asynchronous; document construction either results exclusively from collective decisions or results from both individual and collective initiatives. At least termination is a collective decision.

Solution

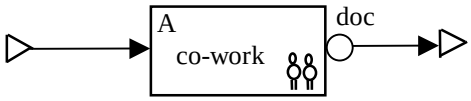


Figure 10.

4.2. The ‘delegated work’ pattern

Problem Collaborative production of a single document in isolation.
Context The group delegates the production work to a single actor and performs a collective review afterwards. A defect list is produced by the review. A rework task follows, which will be defined, if necessary, after the review. The decision to perform or not some rework is taken collectively at the end of the review. There are many ways for structuring the collective review task: for instance, with a single co-work pattern, or with patterns reflecting structured inspection methods (e.g. Fagan [7] or Humphrey [11] methods). We have no room for describing all these specialized patterns here.

Solution

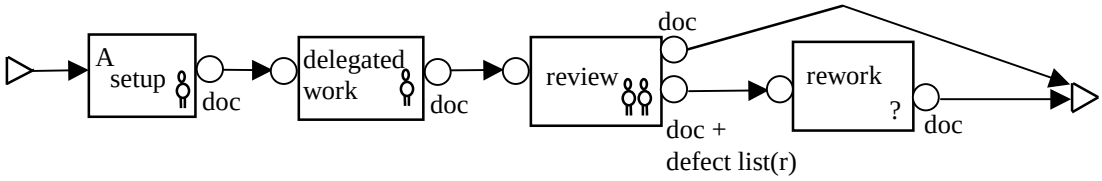


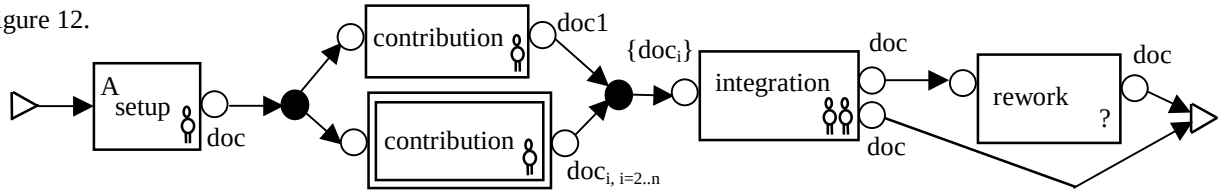
Figure 11.

4.3. The ‘multiprocessing’ pattern

Problem Collaborative production of a single document in isolation.
Context Several actors build concurrently their own version of the same document. They have no access to the contributions from the others. Then, these personal alternative versions are integrated. Such a pattern is often useful at the very beginning of a complex task, when different expertise are required. There are many ways for structuring the collective integration task: integrating two lists of elements is different from integrating two complex design schemata. We have no room for describing all these specialized patterns here.

Solution

Figure 12.



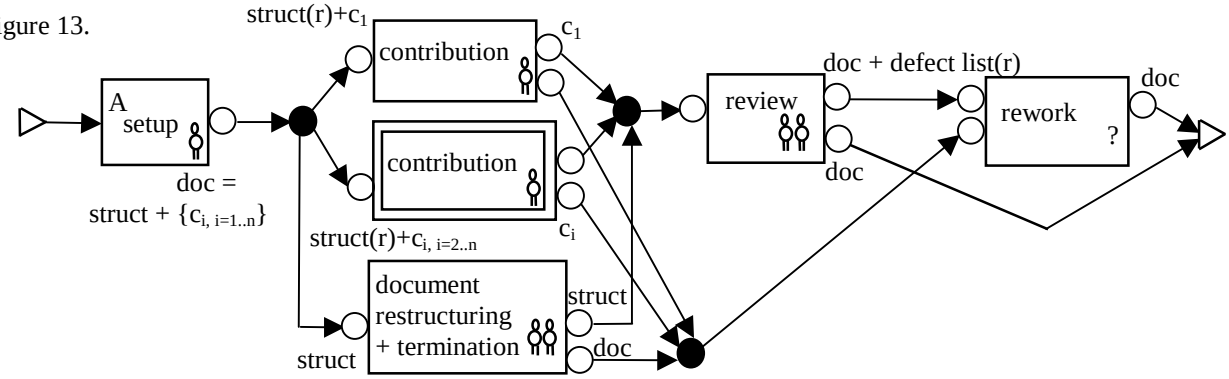
4.4. The ‘division of labour’ pattern

Problem Collaborative production of a single document in isolation.

Context Several actors perform in parallel and independently some partial work for building the document. Their work is mainly independent because it concerns different subparts of the document (named ‘c_i’). However, their work can sometimes interfere, in particular when a modification of the document structuring (named ‘struct’) is required. In this case, a collective document restructuring task creates a new decomposition that will be reworked. This task is also responsible for the collective decision to terminate the parallel work. When termination has occurred, a collective review of the whole document is performed for ensuring its global consistency.

Solution

Figure 13.



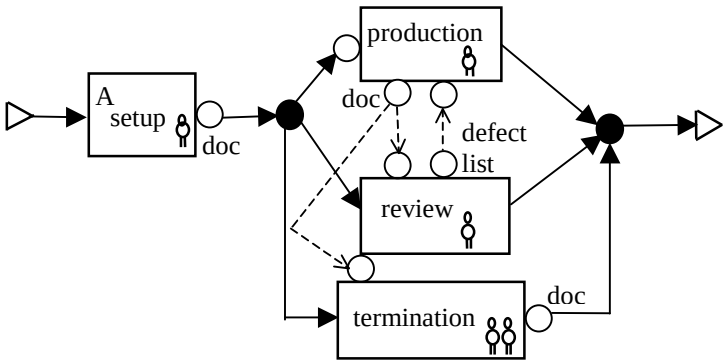
4.5. The ‘producer/reviewer’ pattern

Problem Collaborative production of a single document in isolation.

Context The two concurrent tasks (production and review) exchange products during their execution. There exist many possible policies defining when and by whom products are sent or fetched. The only collective work is about the consensual termination. A similar pattern is described in [10], with no collective task but a distributed termination protocol ensuring that the writer and the reviewer have read the last version of the artifact produced by the other actor.

Solution

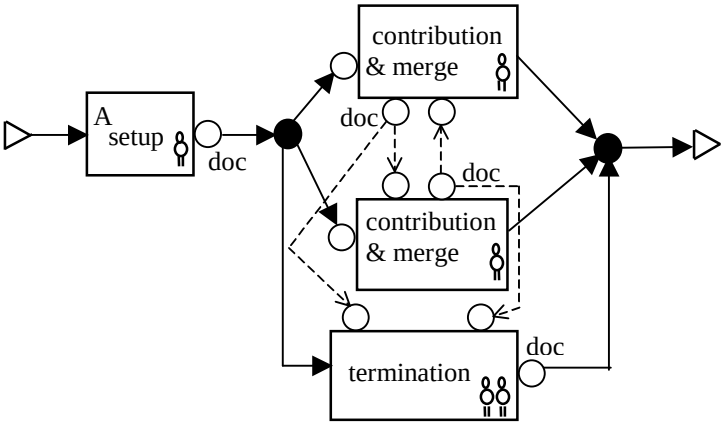
Figure 14.



4.6. The ‘distributed merge’ pattern

Problem Collaborative production of a single document in isolation.
Context Two actors change the same product independently and synchronize it with the other contribution. Different policies may apply for deciding when the document is copied or fetched. Termination results from a collective decision. This pattern becomes complex if more than two actors are involved. A similar pattern with a distributed termination protocol is described in [10].
Solution

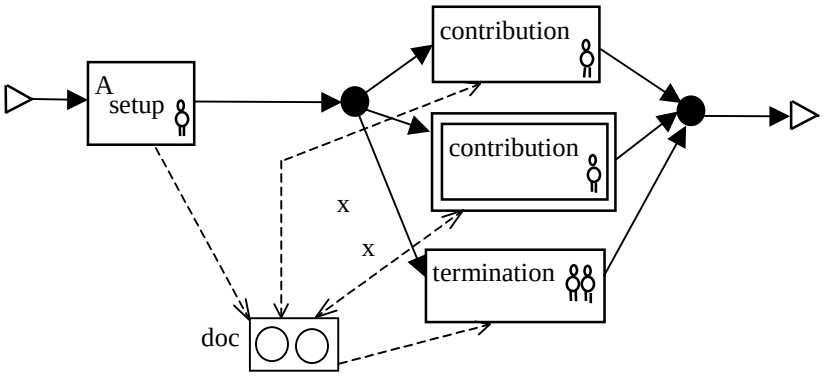
Figure 15.



4.7. The ‘mutual exclusion’ pattern

Problem Collaborative production of a single document in isolation.
Context The document is shared by several tasks. Product transfer into the workspaces is performed through check out/check in operations. The actors work in mutual exclusion because some locking mechanism prevents from concurrent work. Co-work only takes place for defining a consensual termination state.
Solution

Figure 16.



4.8. The ‘collective merge’ pattern

Problem Collaborative production of a single document in isolation.
Context Several actors build the same document. Concurrent work is made possible by managing alternative versions of the document within the repository. An alternative version is created at check in if the checked out version in the repository has already one successor. Merge is performed collectively by the contributors, at will. This task is also responsible for defining a consensual termination state.

Solution

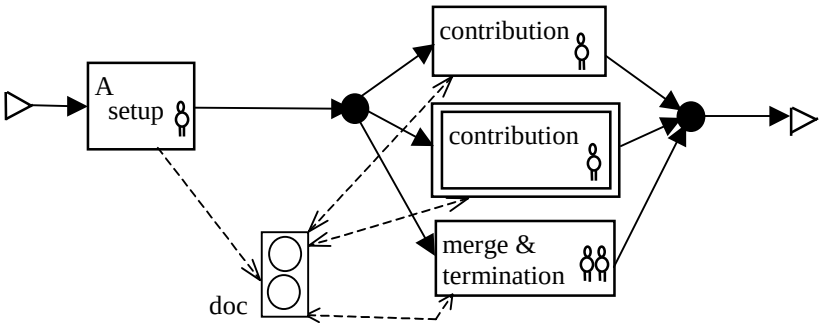


Figure 17.

4.9. The ‘check in and merge’ pattern

Problem Collaborative production of a single document in isolation.

Context In this pattern, merge is performed individually by the contributors. Instead of creating an alternative version in the repository, the actor who has attempted a conflicting check in operation has the responsibility for merging the current version in the repository and his/her version in the workspace [14]. Co-work only takes place for defining a consensual termination state. If some notification mechanism exists the individual merge activity may start as soon as the check in operation concerning the checked out version has occurred.

Solution

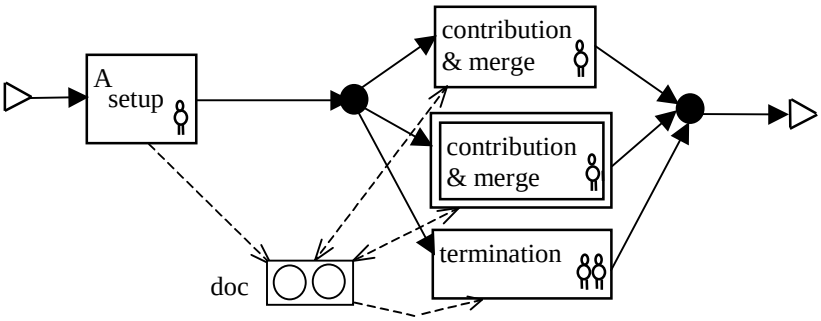


Figure 18.

4.10. The ‘distributed synchronization’ pattern

Problem Synchronization of two dependent documents.

Context Two actors develop their own document and synchronize it with the other document. They exchange their respective documents. Different policies may apply for deciding when the documents are copied or fetched. Termination is the only collective task. The two documents are input parameters of the pattern.

Solution

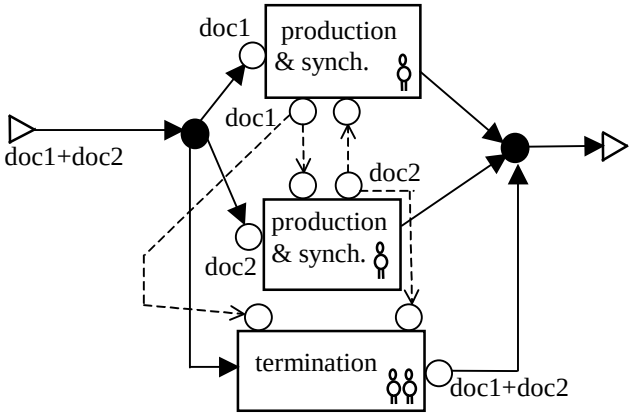
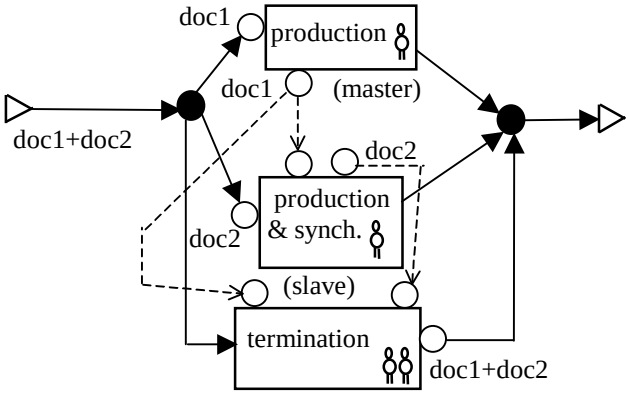


Figure 19.

4.11. The ‘master/slave’ pattern

Problem Synchronization of two dependent documents.
Context Only one actor (the slave) has to synchronize his (her) document with the master’s one. The document can be sent at the master’s initiative or fetched at the slave’s initiative. A similar pattern is described in [10]: the slave can terminate if and only if he/she has read the last version.
Solution

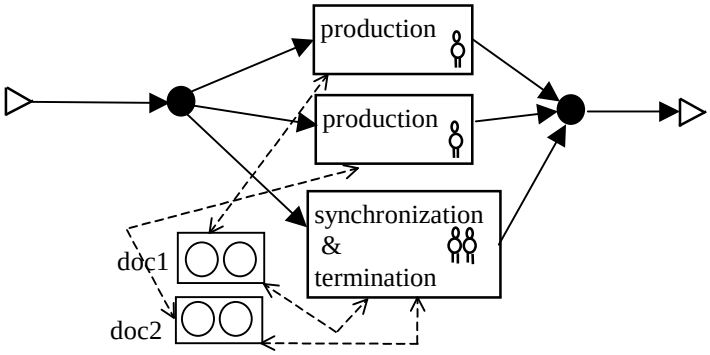
Figure 20.



4.12. The ‘collective synchronization’ pattern

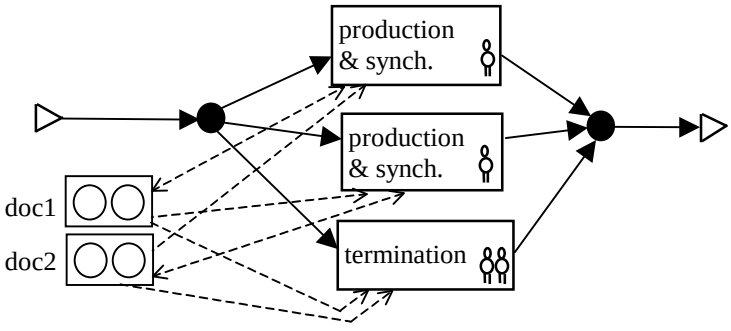
Problem Synchronization of two dependent documents.
Context The documents are shared and their synchronization is performed collectively, at will.

Figure 21.



4.13. The ‘individual synchronization’ pattern

Problem Synchronization of two dependent documents.
Context Synchronization is left to the independent initiative of each actor within his/her individual task. The collective task just ensures that termination is decided collectively.
Solution



4. Conclusion

This paper briefly describes some typical cooperation patterns. On this basis, several implications for PCSs may be emphasized:

1. A PCS should manage individual and collective workspaces, either transactional or cooperative. Product artifacts either flow between them or are shared by them, if there exists some (centralized or distributed) common repository.
2. In order to provide an efficient support for collaborative work, it is necessary to study in depth and to support different kinds of collective conflict resolution tasks, such as review, merge, and synchronization. CSCW approaches [3] are well suited for supporting these tasks, and also for providing direct communication through messages (such as notifications, requests, comments), in addition to indirect communication through product artifacts.
3. Incremental construction of process models is very important, as exemplified by the rework task in several patterns. Integrating an unspecified rework task constitutes a more flexible solution than a schema with a backward iteration: rework can be organized differently than the initial work and the decision is taken dynamically, on the basis of the actual process performance.
4. There exists also a process describing how the process model is built and how it evolves dynamically, ie, a 'meta' process. It is either an individual or a collective process, and PCSs should support it, through process modelling, as the 'production' process is. Specific patterns for structuring the meta process model should exist, as suggested by the loop structure of conversation based workflow systems (see section 2.1).

It would be also very interesting to specify what a well-formed building block is. From the control perspective, some results exist for characterizing well-formed networks of tasks (e.g. [15, 18]), based on the analysis of the corresponding Petri nets. From the collaboration point of view, a first idea could be to distinguish three phases within well-formed collaborative patterns:

- a 'production phase', which must be a well-formed network of tasks,
- a 'collaborative evaluation and/or termination phase', which should always terminate the 'production phase',
- and, possibly, an unspecified 'rework phase' for the dynamic restructuring of the production process, when the evaluation is negative.

All the patterns described in this paper are well-formed from this point of view.

References

[1] ALEXANDER, C., A Pattern Language: Towns, Buildings, Construction. Oxford University Press, New York, 1977.

[2] BANDINELLI, S., FUGGETTA, A., GHEZZI, C., Software Process Model Evolution in the SPADE Environment, IEEE Transactions on Software Engineering, 19, 12, pp 1128-1144, 1993.

[3] ELLIS, C., GIBBS, S., REIN, G., Groupware: Some Issues and Experiences, Communications of the ACM, 34, 1, pp 38-58, 1991.

[4] ELLIS, C., NUTT, G., Modeling and Enactment of Workflow Systems, in: Application and Theory of Petri Nets, Lecture Notes in Computer Science 691, Springer-Verlag, pp 1-16, 1993.

[5] ELMAGARMID, A. (ed.), Database Transaction Models for Advanced Applications, Morgan Kaufmann Pub., 1992.

[6] ESTUBLIER, J., DAMI, S., AMIOUR, M., High Level Process Modelling for SCM Systems, in*:* Proceedings of SCM Workshop, LNCS 1235, Springer-Verlag, pp 81-97, 1997.

[7] FAGAN, M., Design and Code Inspections to Reduce Errors in Program Developments, IBM System Journal, 15, 3, 1976.

[8] FINKELSTEIN, A., KRAMER, J., NUSEIBEH, B. (ed.), Software Process Modelling and Technology, Research Studies Press, 1994.

[9] GAMMA, E., HELM, R., JOHNSON, R., VILLISIDES, J., Design Patterns: Elements of Reusable Object Oriented Software, Addison-Wesley, 1994.

[10] GODART, C., CANALS, G., CHAROY, F., MOLLI, P., SKAF, H., Designing and Implementing COO: Design Process, Architectural Style, Lessons Learned, in: Proceedings of Int. Conference on Software Engineering (ICSE'18), IEEE Press, 1996.

[11] HUMPHREY, W., Managing the Software Process, Addison-Wesley, 1989.

[12] MACDONALD, F., MILLER, J., Automatic Generic Support for Software Inspection, Technical Report RR-96-198, University of Strathclyde, Glasgow, 1996.

[13] MEDINA-MORA, R., WINOGRAD, T., FLORES, R., FLORES, F. The Action Workflow Approach to Workflow Management Technology, in: Proceedings ACM Conference on Computer-Supported Cooperative Work (1992) pp 281-288.

[14] MILLER, T., Configuration management with the NSE, in: Proceedings of Int. Workshop on Software Engineering Environments, LNCS 467, pp 99-106, Springer-Verlag, 1989.

[15] STRAUB, P., HURTADO, C., Understanding Behavior of Business Process Models, in: Proceedings of the First International Conference on Coordination Models and Languages, Cesena, pp 440-443, 1996.

[16] SWENSON, K., MAXWELL, R., MATSUMOTO, T., SAGHARI, B., IRWIN, K. A Business Process Environment Supporting Collaborative Planning, Collaborative Computing, 1 (1994) pp 15-34.

[17] TJAHJONO, D., Building Software Review Systems using CSRS, Technical Report ICS-TR-95-06, University of Hawai, Honolulu, 1995.

[18] VAN DER AALST, W., Verification of Workflow Nets, in: Proceedings of Application and Theory of Petri Nets (ICATPN'97), LNCS 1248, Springer-Verlag, 1997.

[19] Workflow Management Coalition Terminology and Glossary, Technical Report WFMC-TC-1011, 1996.